Sistemas Basados en Microprocesador

Integración y desarrollo de una aplicación: Controlador de un detector de aceleración

Alumnos:
A: Aitor Casado de la Fuente
B: Álvaro Salvador Ruiz

Puesto Nº: x

2023-24

Departamento de Ingeniería Telemática y Electrónica
Universidad Politécnica de Madrid

Índice del documento

1 OBJETIVOS DE LA PRÁCTICA

1.1 Resumen de los objetivos de la práctica realizada

Los alumnos realizadores de dicha memoria, consideran haber adquirido distintas competencias elaborando este proyecto en el cual se desarrolla un sistema de detección de aceleración automático. La propia implementación de distintos módulos en un módulo principal comunicados entre ellos mediante colas permite adquirir una visión mucho más amplia de lo que a programación modular se refiere ya que todo ha de estar perfectamente ordenado y ha de ser legible para poder una mejor comprensión del código.

Este tipo de proyectos permiten adquirir una serie de destrezas y competencias para el desarrollo de software embebido en lenguaje de C++, que permitirá tener una sólida base para el estudiante en cuanto a competencias profesionales se refiere. Además, la integración de hardware y software permite colocarse al estudiante en un proyecto real que puede llegar a desarrollar en algún momento de su vida profesional.

La comprensión y el control de los distintos protocolos de comunicación empleados en este proyecto permite obtener una visión mucho más amplia de lo que es la electrónica y la forma que tienen de entenderse y comunicarse los distintos componentes electrónicos que incluye este proyecto, como por ejemplo I2C, SPI, UART. El manejo de estos protocolos de comunicación es esencial para poder llevar a cabo un correcto funcionamiento del proyecto.

Los estudiantes han desarrollado también la capacidad de búsqueda de distintas fuentes de información para poder tener una mayor visión y comprensión de los distintos módulos con los que han estado trabajando, bien porque han necesitado tener más información que la aportada en la asignatura o bien porque así lo han requerido en el desarrollo del módulo implicado. Además, se ha utilizado un sensor (MPU-6050) del cuál no se tenía ningún tipo de información en cuanto a su funcionamiento por lo que ha sido necesaria la investigación y entendimiento de su funcionamiento para poder integrarlo correctamente en dicho proyecto. El análisis, síntesis y comprensión de las hojas de características (datasheets) de los dispositivos electrónicos es clave para un estudiante de ingeniería electrónica ya que será necesario aplicarlo en un futuro para los distintos proyectos de ingeniería que este aborde.

La validación del funcionamiento de un proyecto tan complejo es clave para poder solucionar posibles errores en el código que a simple vista no se aprecian. Adquirir destreza en la resolución de errores mediante herramientas de depuración es algo básico en la formación de un ingeniero, ya que le permitirá encontrar fallos que supongan un problema mayor para el funcionamiento de la aplicación llevada a cabo.

También se ha empleado instrumentación electrónica y sistemas de medida, así como multímetros y osciloscopios teniendo así un mejor desarrollo en el proyecto.

Realizar este tipo de proyectos obliga a los estudiantes a tener mayor capacidad de organización del código, planificación del tiempo como si de un proyecto real se tratase, así como la toma de decisiones de las diferentes ideas para poder seleccionar mejores alternativas. Además, permite a los estudiantes tener una primera toma de contacto con proyectos que pueden desarrollarse en el campo de la domótica, así como en el campo de IoT, permitiendo obtener experiencia que podrá aplicar en el ámbito profesional.

1.2 Acrónimos utilizados

Identifique los acrónimos usados en su documento.

UART	Universal Asynchronous Receiver Transmitter
I2C	Inter-Integrated Circuit
SPI	Serial Peripheral Interface
LCD	Liquid Crystal Display
CMSIS	Cortex Microcontroller Software Interface Standard
RTOSv2	Real Time Operating System Version 2
IoT	Internet of Things
CAD	Computer-Aided Design
TIM	Timer
SDA	Serial Data
SCL	Serial Clock
MOSI	Master Out, Slave In
SCK	Serial Clock
X	Aceleración del Eje X
Y	Aceleración del Eje Y
Z	Aceleración del Eje Z
refEjeX	Valor de aceleración de referencia del Eje X
refEjeY	Valor de aceleración de referencia del Eje Y
refEjeZ	Valor de aceleración de referencia del Eje Z
LD1	Led verde de la tarjeta núcleo
LD2	Led azul de la tarjeta núcleo
LD3	Led rojo de la tarjeta núcleo

1.3 Tiempo empleado en la realización de la práctica.

Los estudiantes empezaron a desarrollar dicho proyecto a partir del martes 26 de noviembre, fecha en la que tuvo lugar la prueba del segundo bloque de la asignatura. La primera semana se dedicó al desarrollo de los módulos de la hora y el joystick del proyecto, dedicando un total de 7 horas. Durante el primer fin de semana, los estudiantes se dedicaron a comenzar el desarrollo de los módulos del LCD y el Acelerómetro dedicando un total de 8 horas en total durante todo el fin de semana. En la primera semana dedicó un total de 15 horas. En la segunda semana, los estudiantes dedicaron 11 horas con el fin de conseguir implementar los módulos del LCD, Acelerómetro y Leds del proyecto. Durante el fin de semana los estudiantes terminaron de implementar el correcto funcionamiento de los módulos anteriormente mencionados y comenzaron a preparar el módulo del Com_PC del proyecto, dedicando un total de 8 horas. Durante la tercera semana, los estudiantes dedicaron un total de 7 horas para finalizar así la implementación de este módulo y además empezaron a realizar el hilo principal por separado. Durante la última semana antes de la entrega del proyecto final, el número de horas dedicas aumentó entorno a 6, ya que durante el fin de semana, las horas empleadas para el desarrollo del principal pasó a 8 y durante el resto de la semana, de lunes a miércoles, se dedicó un total de 7 horas, siendo así la suma total de horas empleadas de 62 horas.

[Tiempo empleado para realizar la práctica]: El tiempo total empleado ha sido de 62 horas.

1.4 **Bibliografía utilizada**

[RD1] PDF's proporcionados en el Moodle de la asignatura, tanto de los tres bloques de teoría como cada práctica de laboratorio.

[RD2] STM32F429XX Datasheet

[RD3] C. I. D. Novello. *Mastering STM32*. 2015: https://www.embedic.com/uploads/files/20201008/Mastering%20STM32.pdf

[RD4] Nucleo-F429ZI-User Manual PDF

[RD5] STM32F4 HAL Description

[RD6] Página Web del controlador CMSIS RTOS V2: CMSIS: Introduction (arm-software.github.io)

[RD7] GitHub Repository Embedded Libraries: https://github.com/alxhoff/Embedded-libraries/tree/master/STM32/Tested%20and%20working/BH1750

1.5 **Autoevaluación.**

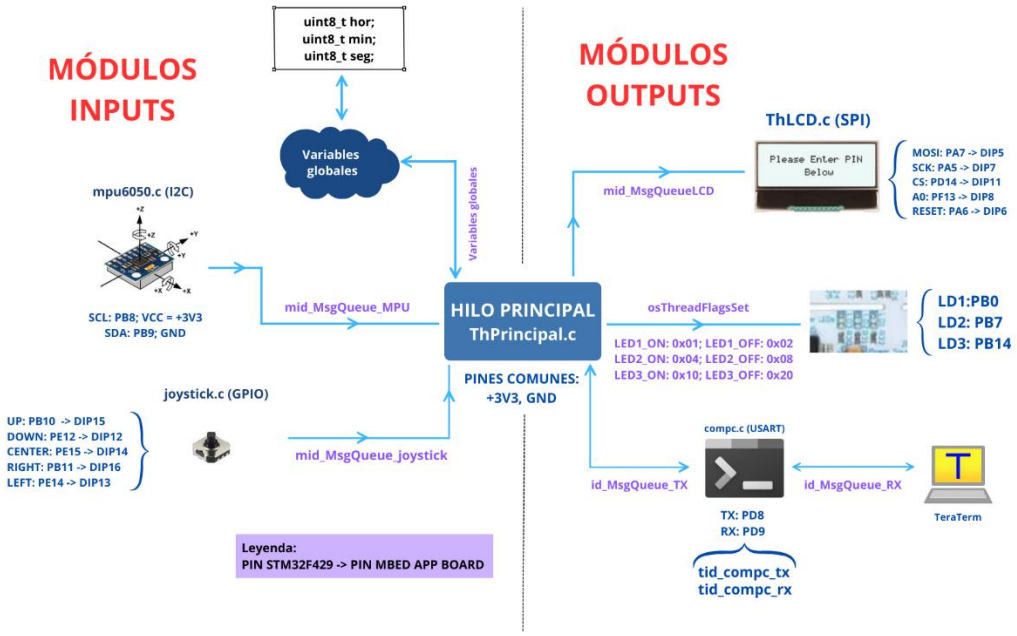
Los alumnos, en base a los resultados de aprendizaje indicados en la guía de la asignatura, consideran que han desarrollado todos los resultados de aprendizaje, aunque no en la misma proporción. Han escrito el código necesario para desarrollar una aplicación basada en microprocesador (RA737). Manejo de entornos CAD para (Keil uVision) para la codificación, compilación y depuración de errores (RA734). Han manejado temporizadores hardware y software para gestionar la temporización y sincronización del proyecto (RA971). Han establecido y gestionado comunicaciones entre dos sistemas empleando diferentes interfaces (RA970) así como conectando periféricos a un microcontrolador empleando interfaces con protocolos estándar como I2C o SPI (RA730).

En base a este desarrollo, los alumnos han adquirido conocimientos de la asignatura. No obstante, los alumnos consideran que hay factores que requieren mejorar como mayor profundidad en la depuración de errores, así como abstracción de código, pudiendo ampliar sus conceptos en este lenguaje de programación.

2 RECURSOS UTILIZADOS DEL MICROCONTROLADOR

2.1 Diagrama de bloques hardware del sistema.

Este apartado debe contener un diagrama de bloques donde se identifiquen claramente los elementos utilizados en el diseño o ejercicio. Debe elaborar una figura que muestre todos los elementos utilizados de la tarjeta NUCLEO STM32F429ZI y su interconexión con los elementos externos (tarjeta de aplicaciones, sensores, etc.).



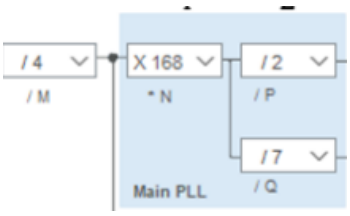
2.2 Cálculos realizados y justificación de la solución adoptada.

En este punto debe describir como ha configurado cada uno de los recursos del microcontrolador, los cálculos que haya realizado y los valores programados en los registros más significativos.

1. Tick del sistema:
El tick del sistema (SYSCLK) se ha configurado para una frecuencia de 168 MHZ, aplicando la siguiente fórmula:

$$PLLCLK = \frac{HSE}{4} * \frac{168}{2} \rightarrow HSE = 8\,MHz \rightarrow PLLCLK = 168\,MHZ \rightarrow SYSCLK = 168\,MHz$$

Para ello, se ha realizado la siguiente configuración en el fichero main.c:




```
RCC_OscInitStruct.PLL.PLLM = 4;
RCC_OscInitStruct.PLL.PLLN = 168;
RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
RCC_OscInitStruct.PLL.PLLO = 7;
```

2.3 **Pines empleados en el proyecto.**

En la siguiente captura, se muestra los pines hardware que se han empleado en dicho proyecto, así como los pines utilizados del acelerómetro:

CONEXIONES STM32 - MBED APPLICATION BOARD

APP.	TIPO	STM32	MBED
SUPPLY	VCC (3.3 V)	CN8.7	DIP40
	GND (0 V)	CN8.11	DIP1
SPI	MOSI	PA7	DIP05
	RESET	PA6	DIP06
	SCK	PA5	DIP07
	A0	PF13	DIP08
	CS_N	PD14	DIP11
JOYSTICK	UP	PB10	DIP15
	RIGHT	PB11	DIP16
	DOWN	PE12	DIP12
	LEFT	PE14	DIP13
	CENTRE	PE15	DIP14
I2C	SCL	PB8	SCL MPU
	SDA	PB9	SDA
USART	USART3_TX	PD8	TX
	USART3_RX	PD9	RX

EXTENSION CONNECTORS

						CN8							CN7				
							1	2	PC8			PC6	1	2	PB8		
					IOREF		3	4	PC9			PB15	3	4	PB9		
					NRST		5	6	PC10			PB13	5	6	AVDD		
					3V3		7	8	PC11			PB12	7	8	GND		
					5V		9	10	PC12			PA15	9	10	PA5		
					GND		11	12	PD2			PC7	11	12	PA6		
					GND		13	14	PG2			PB5	13	14	PA7	PB5	
					VIN		15	16	PG3			PB3	15	16	PD14		
												PA4	17	18	PD15		
						CN9						PB4	19	20	PF12		
					PA3		1	2	PD7								
					PC0		3	4	PD6					CN10			
					PC3		5	6	PD5			AVDD	1	2	PF13		
			PF3	PD11	PC1		7	8	PD4			AGND	3	4	PE9		
		Open: PB9	PF5	PD12	PC4		9	10	PD3			GND	5	6	PE11		
		Open: PB8	PF10	PD13	PC5		11	12	GND			PB1	7	8	PF14		
				PA1	PB2		13	14	PE2			PC2	9	10	PE13		
			COMP2_INP		PB6		15	16	PE4		PA2	PF4	11	12	PF15		
				PF2	PA8		17	18	PE5			PB6	13	14	PG14	PC4	PG9
				PF1	PA9		19	20	PE6			PB2	15	16	PG9	PC5	PG10
				PF0	PA10		21	22	PE3			GND	17	18	PE8		
					GND		23	24	PF8		PF10	PD13	19	20	PE7		
					PD0		25	26	PF7		PF5	PD12	21	22	GND		
					PD1		27	28	PF9		PF3	PD11	23	24	PE10		
					PG0		29	30	PG1			PE2	25	26	PE12		
												GND	27	28	PE14		
												PA0	29	30	PE15		
												PB0	31	32	PB10		
												PE0	33	34	PB11		

3 SOFTWARE

3.1 Descripción de cada uno de los módulos del sistema

3.1.1 Módulo HORA:

Este módulo es el encargado de llevar a cabo la cuenta del reloj del sistema de control, indicando el usuario cuánto tiempo lleva activo el sistema de control y así poder representarlo constantemente.

Entradas: Ninguna.
Salidas: variables globales: uint8_t horas, uint8_t minutos y uint8_t segundos
Método de sincronización: Variables globales

Ficheros: hora.c y hora.h

Funciones que implementa:

- **int Init_Hora (void):** función en la que inicializamos tanto el timer de 1 segundo, como el hilo del reloj.
- **void Segundero_Callback(void const *arg):** lleva a cabo la rutina del timer a la cual se entra cada segundo. Dentro de ella hay un código con los tres contadores de la hora (horas, minutos y segundos), los cuáles se gestionan como el funcionamiento de un propio reloj.
- **void ThClock (void *argument):** función que ejecuta el código del hilo. En este caso únicamente arranca el timer de 1 s.

3.1.2 Módulo joystick:

Este módulo se encarga de **interpretar y gestionar todas las interacciones del usuario con el joystick**. Para ello, utiliza colas de mensajes para almacenar información sobre cada pulsación detectada. Cada mensaje contiene detalles como la dirección de la pulsación, su duración y si se trata de una pulsación corta o larga. El módulo también dispone de diversas funciones internas que permiten identificar el tipo de pulsación, determinar la dirección de pulsación, almacenar la información de la pulsación en una cola de mensajes y mandarla al posterior hilo principal.

Entradas: tipo de pulsación realizada (Hardware Input)
Salidas: mensaje en función de la pulsación realizada, msg_enviado.pulsacion
Método de sincronización: cola de mensajes, mid_MsgQueueJOY

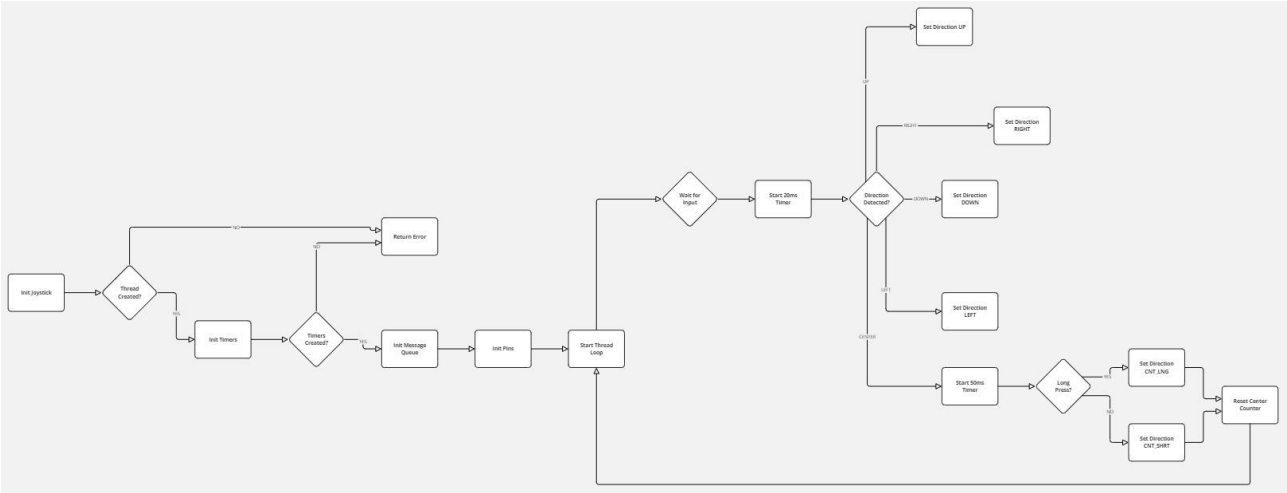
Ficheros: joystick.c y joystick.h

Funciones que implementa:

- **int Init_Joystick (void):** función que inicializa el hilo
- **void ThJoystick (void *argument):** recibe un flag de interrupción de un joystick
- **void Init_Pines (void):** inicializa los puertos de los pines del joystick, además de llamar a la rutina de interrupción del joystick
- **void Timer_Callback_20ms (void const *arg):** gestiona el tipo de pulsación del joystick, identificando la dirección de esta. Además, inicia el segundo timer que llevará a cabo la cuenta de la duración de la pulsación.
- **void Timer_Callback_50ms (void const *arg):** gestiona la pulsación del botón CENTER del joystick, identificando además si esta ha sido una pulsación corta o larga
- **void EXTI15_10_IRQHandler(void):** rutina de interrupción del joystick que indica el tipo de pulsación realizada. Se encuentra en el archivo de interrupciones stm32f4xx_it

- **void HAL_GPIO_EXTI_Callback(uint16_t GPIO_PIN):** gestiona la rutina de interrupción, iniciando un flag que indica que se ha pulsado el joystick en cualquier dirección. Se encuentra en el archivo de interrupciones stm32f4xx_it.

El siguiente diagrama muestra el funcionamiento de dicho módulo:



3.1.3. Módulo Acelerómetro MPU6050

Este módulo está implementado para permitir la comunicación entre el acelerómetro y la tarjeta núcleo. Concretamente, gestiona mediante el protocolo I2C la realización de 4 medidas en tiempo real: aceleración en eje X, aceleración en el eje Y, aceleración en el eje Z y por último la medida de la temperatura. Cada segundo, tendrá que ir transmitiendo dichas medidas a una cola de mensajes que enviará al módulo principal, y este se encargará de enviarla al módulo del lcd para su representación en el display.

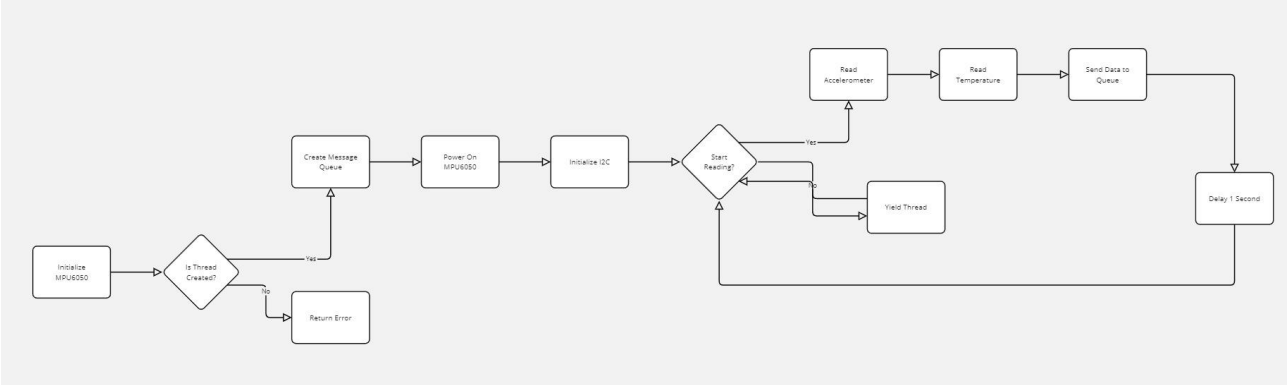
Entradas: aceleración y temperatura medidas
Salidas: aceleración y temperatura enviadas al hilo principal
Método de sincronización: cola de mensajes, mid_MsgQueue_MPU

Ficheros: mpu6050.c y mpu6050.h

Funciones que implementa:

- **void Thread_MPU6050 (void *argument):** implementa el funcionamiento del hilo, inicializa al sensor y envía por la cola de mensajes la aceleración y la temperatura.
- **void Callback_I2C (uint32_t event):** función que gestiona la transmisión del bus I2C. Envía un flag cuando se completa una transferencia.
- **void PWR_ON_MPU6050 (void):** función que inicializa al bus I2C, configura los registros de inicialización del sensor y lo prepara para la medida, configurada a +/- 2 g. Primero, despierta al sensor enviando un 0x00 al registro 0x6B. Después configura una frecuencia de muestreo enviando al registro 0x19 un 0x00. Posteriormente, configura la medida a +/-2 g enviando un 0x00 al registro 0x1C.
- **static float RD_ACCEL_MPU (uint8_t reg_address):** esta función realiza la medida de la aceleración del sensor. Como parámetro se le pasa el registro del que queremos leer. Dentro de la función, lee primero la parte alta de la aceleración y posteriormente se aumenta en uno el registro a leer para así leer la parte baja de la aceleración. A esta función se le llama tres veces para realizar la medida en cada eje. Internamente, hay que tratar la información que nos llega. Para ello, los 8 primeros bits que llegan se desplazan y después de concatenan los bits de la parte baja, para terminar, dividiéndolo entre 16384 que es la resolución que indica el fabricante. La dirección de los registros se define como macros: ACCEL_XOUT_H: 0x3B; ACCEL_YOUT_H: 0x3D; ACCEL_ZOUT_H: 0x3F.

- **static float RD_TEMP_MPU (uint8_t reg_address):** esta función realiza la medida de la temperatura del sensor. Como parámetro, se le pasa el registro del que vamos a leer la temperatura. Al igual que antes, del primer registro se leen los 8 bits más altos, e internamente se aumenta en uno la variable que se le pasa como parámetro para leer del segundo registro los bits de la parte baja. Internamente se desplazan los 8 bits más altos y se concatenan los bits más bajos. Eso se divide entre 340 y se le acaba sumando 36.53, que es la función de transferencia que indica el fabricante. La macro del registro de la temperatura es: TEMP_OUT_H: 0x41.



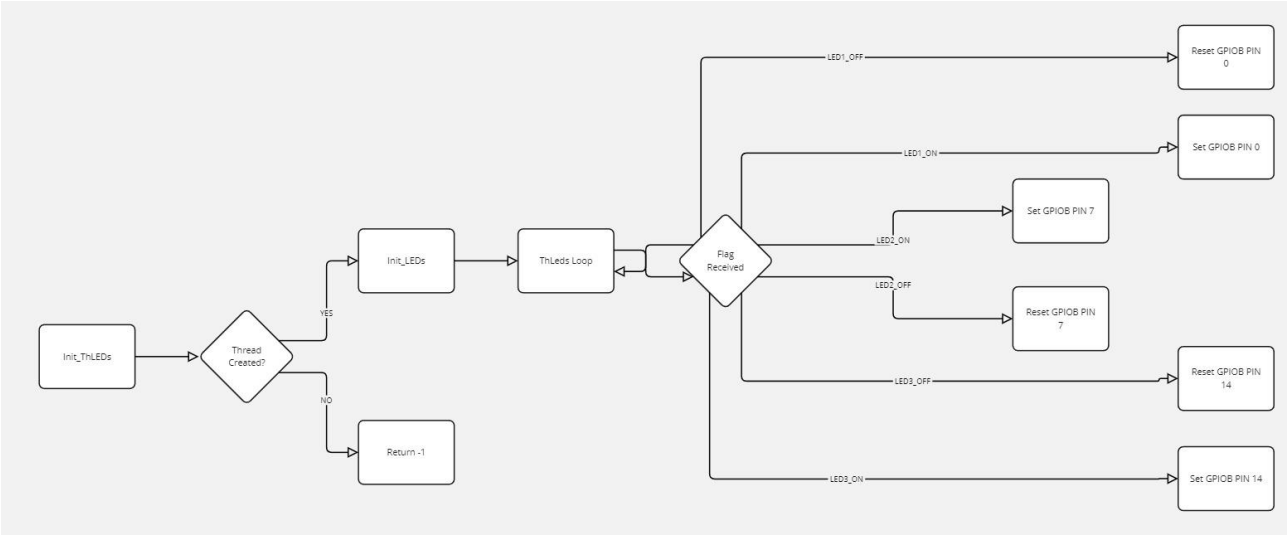
3.1.4. Módulo Leds Tarjeta Núcleo

Este módulo será el encargado de llevar la gestión del estado de los tres leds con los que cuenta la tarjeta núcleo en función de los valores de aceleración y temperatura medidos por el acelerómetro. Para ello, se hará uso de una serie de flags, los cuales serán enviados por el principal una vez se realice la comparación de los valores medidos por el acelerómetro con los valores de referencia, que indicarán el estado correspondiente de cada led.

Entradas: distintos flags en función del resultado de comparar la aceleración y temperatura medidas por el acelerómetro con los valores de referencia correspondientes.
Salidas: Cambio del estado de los leds en función del flag recibido.
Método de sincronización: Flags, LED1_ON, LED1_OFF, LED2_ON, LED2_OFF, LED3_ON, LED3_OFF.

Ficheros: leds_N.c y leds_N.h

Funciones que implementa:
- **int Init_ThLEDs (void):** función en la que se crea el hilo del módulo, y se inicializan los leds.
- **void ThLeds (void *argument):** función principal del hilo, inicializa el hilo y espera a recibir cualquier flag para así actualizar el estado del led correspondiente.
- **void Init_LEDs (void):** función que inicializa y configura los leds a emplear de la tarjeta núcleo.



3.1.5. Módulo LCD

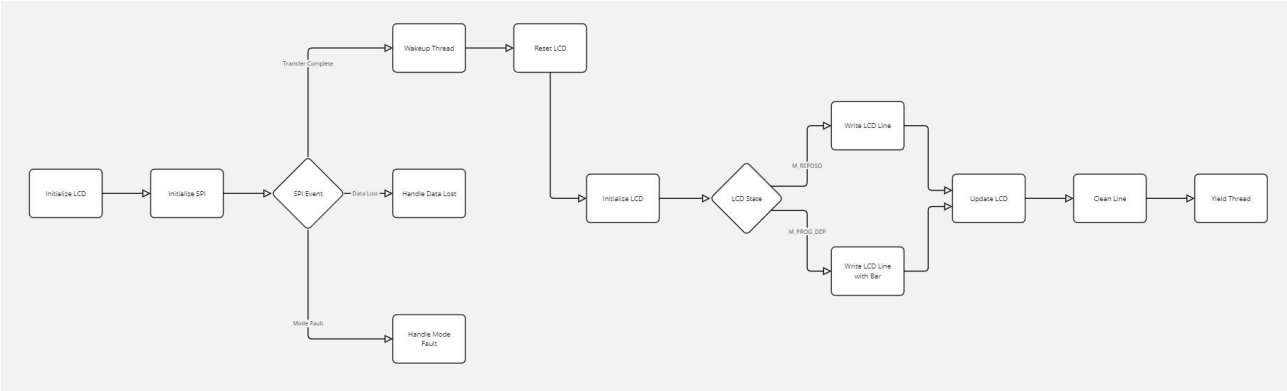
Módulo encargado de gestionar la información que se quiere mostrar en el display del LCD. Para ello, se emplea el bus SPI. Mediante este módulo podremos imprimir por pantalla la información gráfica que el sistema de control quiera mostrar al usuario, indicando el modo en el que se encuentre además de la acción que se esté realizando justo en ese modo.

Entradas: mensaje recibido por la cola del tipo estructura, rxMsg
Salidas: ninguna (información mostrada por el LCD)
Método de sincronización: cola de mensajes, mid_MsgQueueLCD

Ficheros: lcd.c, lcd.h, Arial12x12.h

Funciones que implementa:

- **int Init_LCD (void):** función que inicializa el hilo del LCD
- **void ThLcd (void *argument):** función que implementa el hilo del LCD
- **void SPI_Callback (uint32_t event):** finción que controla los eventos del bus SPI
- **void initPines (void):** inicializa los pines que intervienen en el hilo
- **void delay (uint32_t n_microsegundos):** función que establece un retardo en microsegundos necesaria para el reset
- **void LCD_reset (void):** prepara el LCD para ser inicializado
- **void LCD_wr_data (unsigned char data):** envía un dato al LCD
- **void LCD_wr_cmd (unsigned char cmd):** envía un comando al LCD
- **void LCD_update (void):** actualiza la información representada en el LCD
- **void cleanBuffer (uint8_t line):** función que limpia el buffer del LCD
- **void cleanLine (void):** función que inicializa las líneas de escritura en el LCD
- **void LCD_init (void):** inicializa el LCD
- **void LCD_symbolToLocalBuffer_L1 (uint8_t symbol):** escribe un símbolo del fichero Arial12x12 en la línea 1
- **void LCD_symbolToLocalBuffer_L2 (uint8_t symbol):** escribe un símbolo del fichero Arial12x12 en la línea 2
- **void LCD_symbolToLocalBuffer_L3 (uint8_t symbol):** función empleada para escribir la barra baja que selecciona el carácter a editar en el modo depuración y programación
- **void symbolToLocalBuffer (uint8_t line, uint8_t symbol):** escribe un símbolo Arial 12x12 en la línea que le indicada por parámetro
- **void writeLCD (char *miArray, uint8_t line):** función que escribe en la línea indicada el array que se le indique



3.1.6. Módulo COM-PC

Este módulo será el encargado de gestionar todo lo que tenga que ver con la comunicación que se llevará a cabo entre el micropocesador y el PC mientras que el micro se encuentre en el modo depuración. Para esto, se empleará el bus USART. Mediante este módulo seremos capaces de realizar modificaciones en el valor del tiempo total que lleva el sistema ejecutándose, y variaciones en los valores de referencia con los que contarán los distintos ejes del acelerómetro.

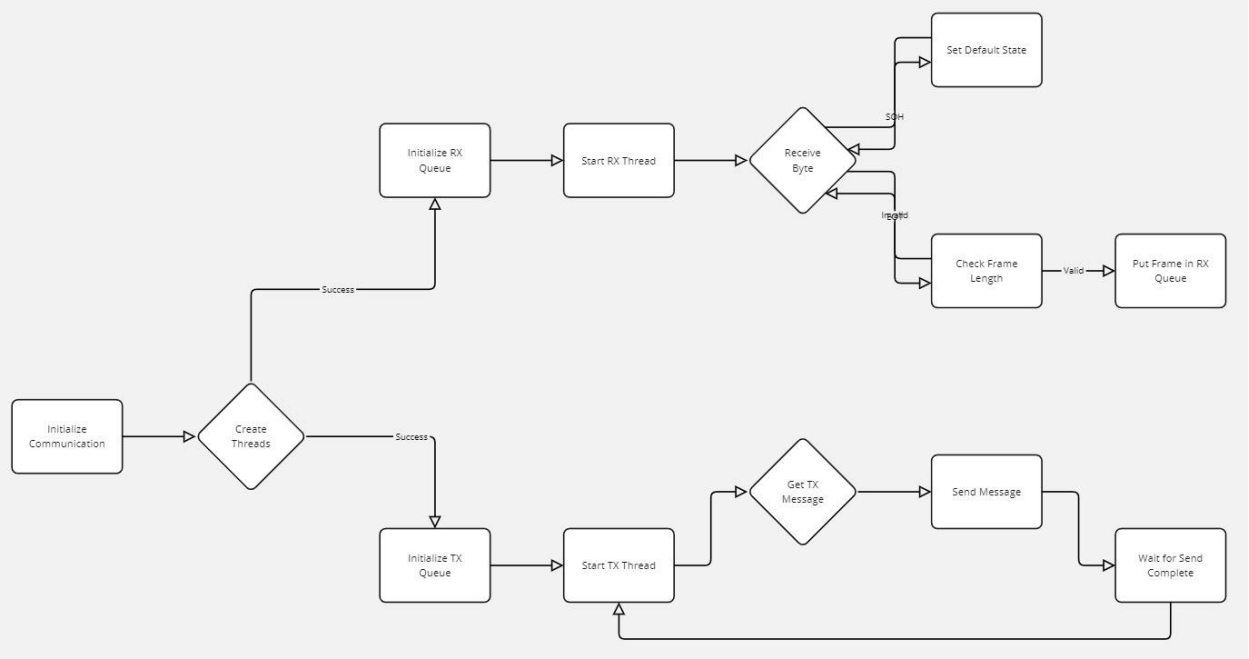
Entradas: tramas recibidas por la cola con identificador id_MsgQueue_RX del PC.
Salidas: tramas enviadas por la cola con identificador id_MsgQueue_RX al Principal.
Método de sincronización: 2 colas de mensajes: id_MsgQueue_RX, id_MsgQueue_TX

Ficheros: compc.c, compc.h

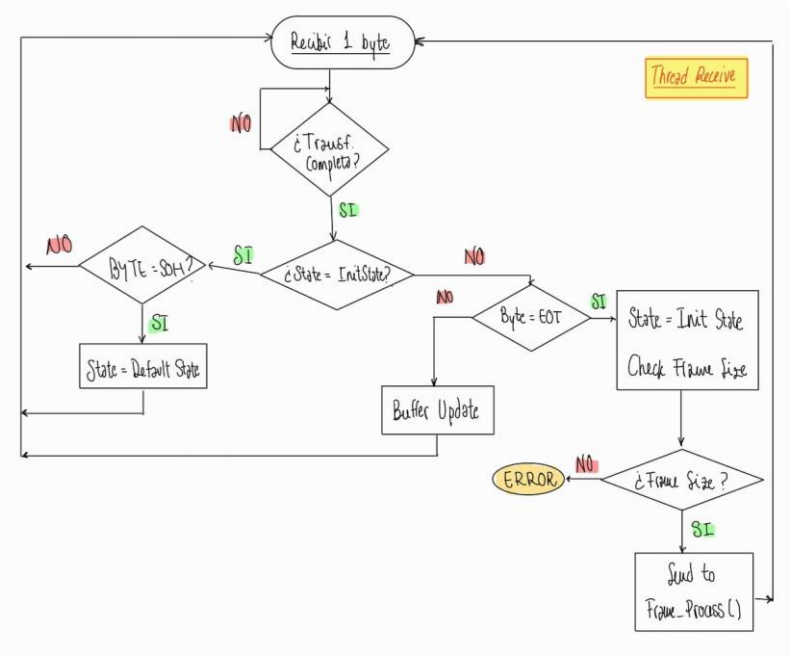
Funciones que implementa:

- **int Init_com (void):** función en la que se crean tanto los dos hilos como las dos colas necesarias para este módulo.
- **void myUART_Thread_tx (void* args):** función principal del hilo de recepción que se encargará de recibir la trama de respuesta enviada por el principal y enviarla al PC.
- **void myUART_Thread_rx (void* args):** función principal del hilo de transmisión que se encargará de recibir la trama enviada por el PC y enviársela al principal.
- **void myUSART_Callback (uint32_t event):** función en la que se gestiona la recepción de los flags correspondientes cuando se realiza tanto una transmisión como una recepción de tramas.
- **osMessageQueueId_t idQueueRX (void):** función que nos devolverá el identificador de la cola empleada para almacenar las tramas recibidas del PC.
- **osMessageQueueId_t idQueueTX (void):** función que nos devolverá el identificar de la cola empleada para la transmisión de las tramas.
- **void processingFrame (uint32_t flagRecepcion):** función en la que procesaremos la trama recibida por el PC antes de enviársela al principal.

Se adjunta un organigrama que desarrolla dicho módulo:



Además, se adjunta también una captura que muestra el autómata cuando se recibe la trama enviada desde el TeraTerm:



3.1.7. Módulo Principal:

Este módulo principal reside en el **núcleo del sistema** y desempeña un papel crucial en la **gestión global** del mismo. Su función principal es la **coordinación** de todos los módulos descritos anteriormente, asegurando una **interacción fluida y eficiente** entre ellos. Gestiona cada módulo y que información se procesa en cada uno, dependiendo de la información recibida de los otros módulos (output) a través de las colas de mensajes. Estas también son usadas para comunicar con los módulos (input), que gestionarán la información enviada desde este hilo principal para lograr la función indicada.

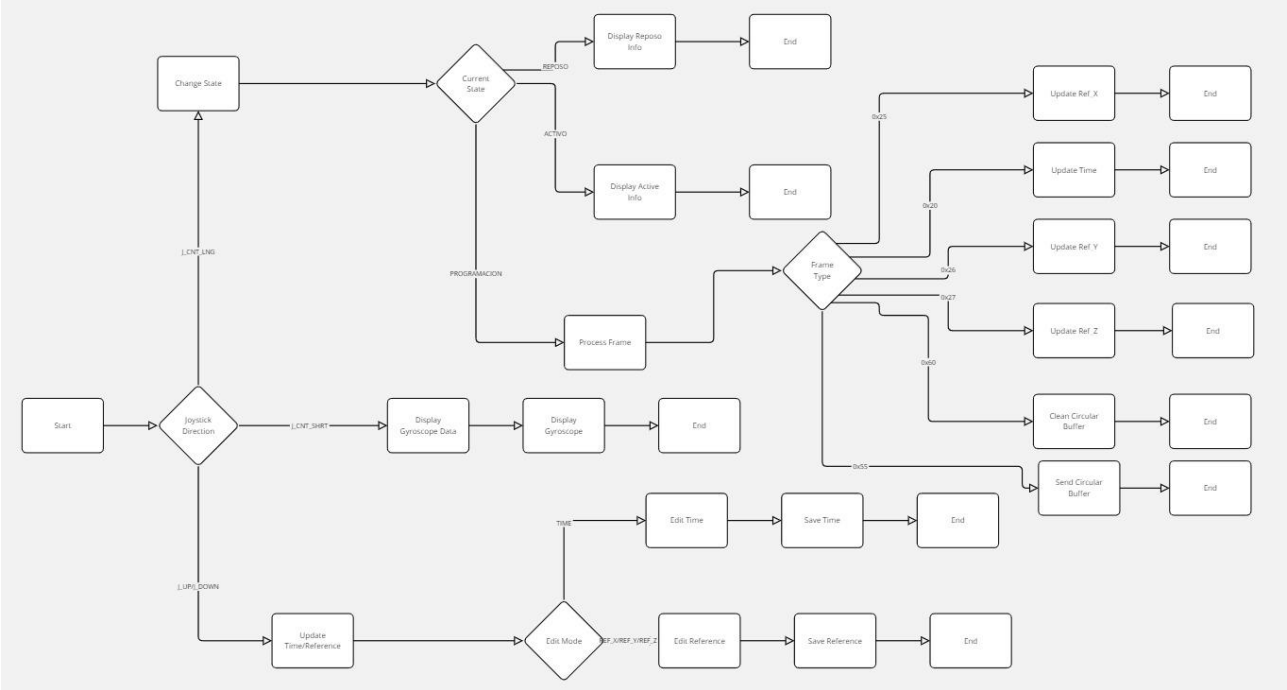
Entradas: mensajes de los módulos INPUT a través de las colas de mensajes
Salidas: mensajes de los módulos OUTPUT a través de las colas de mensajes
Método de sincronización: cola de mensajes de cada módulo

Ficheros: principal.c y principal.h

Funciones que implementa:

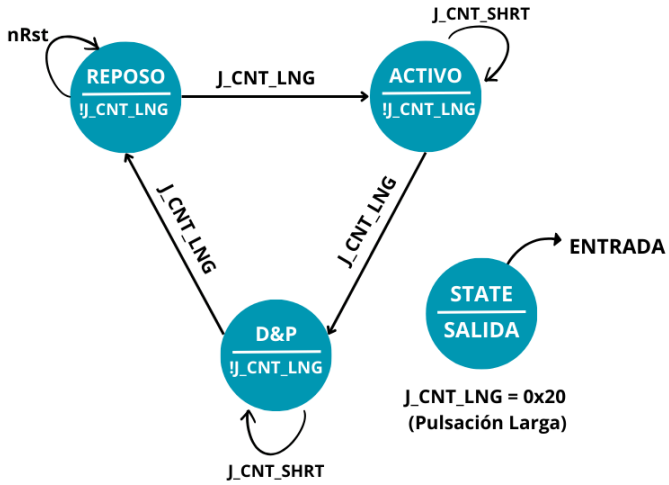
- **int Init_Principal (void):** función en la que creamos el hilo del módulo principal.
- **void principal (void *argument):** función principal del módulo en la que crearemos un autómata para gestionar los distintos estados de este.
- **void infoReposo (**void): función en la que enviamos al LCD el texto a representar cuando nos encontremos en el modo Reposo.
- **void infoProgramacion (void):** función en la que mostraremos por pantalla la información en el modo programación.
- **void encendidoLeds (MSGQUEUE_MPU6050_OBJ_t valores):** función en la que realizaremos la comparación entre los distintos valores de aceleración de cada uno de los ejes medidos por el acelerómetro con los valores de referencia y enviaremos el flag correspondiente al módulo de los Leds para modificar el estado de cada uno de estos.
- **void apagadoLeds (void):** función en la que enviaremos al módulo de los Leds los flags correspondientes para apagar los 3 Leds de la tarjeta Núcleo.
- **void saveCircBuff (MSGQUEUE_MPU6050_OBJ_t buff):** función encargada de ir almacenando en el buffer circular las medidas realizadas por el acelerómetro.
- **void updateTime (MSGQUEUE_JOY_t pulsacion):** función encargada de gestionar el cambio de la hora en el modo programación.
- **void updateReference (MSGQUEUE_JOY_t joystick):** función encargada de gestionar el cambio de cada una de las referencias de cada eje en el modo programación.
- **uint8_t setUnidades (uint8_t uni):** función que nos devuelve el valor de las unidades de la variable que le pasamos como parámetro.
- **uint8_t setDecenas (uint8_t dec):** función que nos devuelve el valor de las decenas de la variable que le pasamos como parámetro.
- **void cleanCircBuff (void):** función empleada para realizar una limpieza de todas las medidas almacenadas en el buffer circular.
- **void sendCircBuff (void):** función empleada para realizar el envío de todas las medidas almacenadas en el buffer circular.
- **void processFrame (char array[]):** función en la que procesaremos cada una de las tramas obtenidas desde el Teraterm

Se adjunta un organigrama que desarrolla dicho módulo:



3.2 Descripción global del funcionamiento de la aplicación. Descripción del autómata con el comportamiento del software (si procede)

Para poder abordar mejor el comportamiento de este sistema basado en un acelerómetro, se decide hacer una descripción de este en base a un autómata implementado en el proyecto. El autómata es el que se muestra en la siguiente figura:



En este autómata, los cambios de estado/modos de funcionamiento se realizan mediante una pulsación larga del botón central del joystick (J_CNT_LNG: 0x20), considerada mayor a 1 segundo. Dentro de cada modo de funcionamiento, a excepción del modo reposo, si se realiza una pulsación corta del botón central (J_CNT_SHRT: 0x10), pasa a realizar las funciones propias de esos modos de funcionamientos, los cuáles se describirán a continuación.

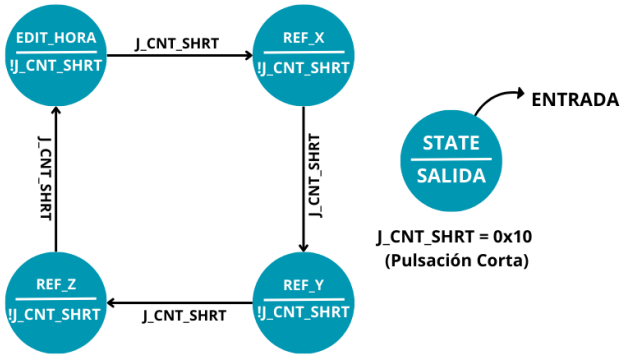
Tras una pulsación del botón reset (reset asíncrono), el sistema comienza en el modo **REPOSO**. En este modo únicamente se muestra por el LCD dos líneas. En la primera "SBM 2024" y en la segunda la hora en el siguiente formato de las horas: HH:MM:SS. En el momento que el programa comienza a funcionar, dicho contador comienza a funcionar. En dicho modo, no se debe recibir ningún estímulo a través de hardware, únicamente una pulsación larga del botón central de joystick para pasar al siguiente modo de funcionamiento. La visualización en el LCD es la siguiente:

Tras una pulsación del botón central del joystick (J_CNT_LNG: 0x20), el sistema pasará automáticamente al modo **ACTIVO**. En dicho modo, entra en funcionamiento el sensor **MPU-5060**. Dicho sensor se trata de un giroscopio de tres ejes. Sus características principales son que la comunicación es a través del protocolo I2C, además de incluir un ADC interno de 16 bits. Como extras, puede proporcionar una temperatura ambiente ya que incluye un sensor de temperatura, además de que es posible configurarlo en modo de bajo consumo. Es posible configurar el rango de aceleración, que en nuestro caso se realiza para ±2g. También es posible medir el rango de giro del osciloscopio. En este modo de funcionamiento, el acelerómetro realiza las medidas de los tres ejes (X, Y y Z) cada segundo, enviándolas a través de una cola de mensajes del hilo principal. Estos tres valores de aceleración son mostrados por la pantalla del LCD nada más ser recibidos. Además, como bien se ha descrito, entrega un valor medido de la temperatura actual, representada también en el LCD. En dicho modo, se fijan valores de referencia de aceleración, los cuales al principio para los tres ejes es el mismo, de +1.0 g (refEjeX, refEjeY y refEjeZ). Además, se implementan los tres leds de la tarjeta núcleo. Cada led está asociado a un valor de aceleración (LD1 = X, LD2 = Y, LD3 = Z). En el momento en el que el valor medido de la aceleración es mayor al valor de referencia fijado, se envía un flag al led correspondiente para encenderse. Mientras todo este proceso se realiza de manera en la que el usuario puede ver lo que está ocurriendo, en segundo plano, las medidas se están almacenando en un buffer circular, el cual almacena las últimas diez medidas de la aceleración realizadas, así como de la hora y temperatura. Dichos valores se almacenan en un array de caracteres de 43 posiciones con la siguiente forma: **HH:MM:SS--Tm:TT.TºC-Ax:n.n-Ay:n.n-Az:n.n.** La utilidad de esto es que en el siguiente modo, se le solicite al sistema la lectura de las medidas mediante una trama, y que estas sean mostradas por pantalla.

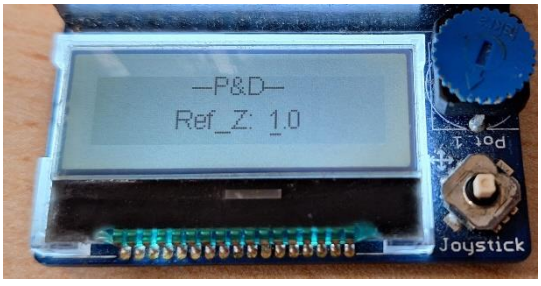
La información que se muestra en el LCD en este modo es la siguiente:

Por último, tras realizar una pulsación larga del botón central del joystick (J_CNT_LNG: 0x20) se cambia al último modo de funcionamiento, **DEPURACIÓN/PROGRAMACIÓN.** El objetivo de este modo no es otro que poder configurar el sistema como desee el usuario, permitiendo variar la hora y las tensiones de referencia como el mismo desee. Hay dos formas de realizar esto: mediante programación externa hardware, es decir

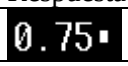
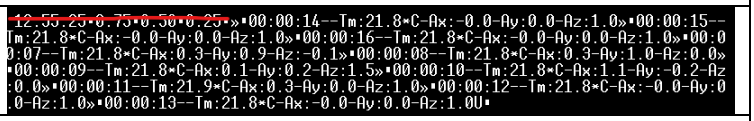
editar los campos con el joystick; y haciendo uso del puerto USART del micro. Con cada pulsación corta del botón central del joystick (J_CNT_SHRT: 0x10), permitirá variar el modo de edición. Dentro de este modo, se hace uso de un pequeño autómata que muestra el modo de funcionamiento de dicho modo:



Dicho autómata muestra que, con cada pulsación corta del joystick, se cambia el modo de edición. Con el resto de los gestos del joystick, se permitirá editar el campo de cada modo de programación. Dependiendo del modo de edición, en el LCD se mostrarán los siguientes campos:



Mediante el protocolo de comunicación USART, se realizará la comunicación entre el PC y el sistema. Se realizan a través del canal RS232, configurado para transmitir a 155200 baudios datos de 8 bits con un bit de stop y sin bit de paridad. Para esta comunicación, se emplea el programa TeraTerm, el cual permite enviar tramas que el sistema detectará para realizar una función dependiendo de la trama enviada. La configuración de las tramas con su respectiva respuesta se muestra en la siguiente tabla:

Trama que cambia la hora del sistema	Respuesta del sistema a la trama de la hora
send $01 $20 $0C "12:55:25" $FE	
Trama que cambia la refEjeX	Respuesta del sistema a la trama referencia eje X
send $01 $25 $08 "0.75" $FE	
Trama que cambia la refEjeX	Respuesta del sistema a la trama referencia eje Y
send $01 $26 $08 "0.50" $FE	
Trama que cambia la refEjeX	Respuesta del sistema a la trama referencia eje Z
send $01 $27 $08 "0.25" $FE	
Trama que muestra todas las medidas del buffer	Respuesta del sistema mostrando todas las medidas
send $01 $55 $04 $FE	
Trama que borra todas las medidas del buffer	Respuesta del sistema tras borrar todas las medidas
send $01 $60 $04 $FE	

3.3 Descripción de las rutinas más significativas que ha implementado.

1. Rutina de atención a la interrupción por una pulsación del joystick:

Mediante esta forma, es posible detectar las pulsaciones del joystick y así poder distinguir el tipo de pulsación realizada mediante interrupciones externas del usuario.

```
void EXTI15_10_IRQHandler(void){ //Rutina de la interrupción del joystick
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_10);//Hacia Arriba
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_11);//Hacia la derecha
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_12);//Hacia Abajo
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_14);//Hacia la Izquierda
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_15);//Hacia el Centro
}

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_PIN){
  osThreadFlagsSet(tid_joystick, PULSACION);

}
```

2. Rutina de atención a la interrupción del sensor MPU6050:

```
//Funcion Callback del I2C
void Callback_I2C (uint32_t event){
  uint32_t mask = ARM_I2C_EVENT_TRANSFER_DONE;
  if(event & mask){
    osThreadFlagsSet(tid_mpu6050, INIT_FLAG);
  }
}

void PWR_ON_MPU6050 (void){

  uint8_t byte[2];

  I2Cdrv->Initialize   (Callback_I2C);
```

Mediante esta función, cuando se completa una transferencia del bus I2C, se envía un flag indicando que la transferencia se ha realizado con éxito. Ese flag, INIT_FLAG (0x01) se espera cada vez que hay un MasterTransmit o un MasterReceive. Es importante aclarar que este tipo de transferencia es no bloqueante, y por tanto no bloqueará al micro durante

3. Rutina de atención a la interrupción del LCD:

Una vez enviada la información al LCD, es necesario esperar a que se realiza una transferencia completa en el bus SPI, gestionándolo con la función Callback correspondiente. Además, cada vez que se desea mostrar información en el LCD, es necesario borrar el buffer donde almacena esta información con de actualizarlo. Es importante aclarar que este tipo de transferencia es no bloqueante, y por tanto no bloqueará al micro durante su ejecución.

```
void SPI_Callback (uint32_t event)
{
    switch (event)
    {
    case ARM_SPI_EVENT_TRANSFER_COMPLETE:
        /* Success: Wakeup Thread */
        osThreadFlagsSet(tid_ThLCD, S_TRANS_DONE_SPI);
        break;
```

```
cleanBuffer (rxMsg.line);
modosPrincipal ();
LCD_update ();
cleanLine ();
```

Cada vez que se realiza una transferencia al bus SPI, tanto con Send como Receive se espera este flag:

```
//Escribimos un dato enviandolo al SPI con ->Send
SPIdrv->Send(&data, sizeof(data));

//Esperamos a que se libere el bus SPI: Diapositiva 10 de CMSIS Driver
osThreadFlagsWait(S_TRANS_DONE_SPI, osFlagsWaitAny, osWaitForever);
```

4. Rutina de atención a la interrupción del COMPC:

Como en las anteriores rutinas de atención a la interrupción, es necesario hacer la espera a que se realicen la transferencia completa, en este caso, tanto de la transmisión como de la recepción. De la misma forma, este tipo de transferencia es no bloqueante, y por lo tanto el micro no se verá bloqueado durante su ejecución. Para esta comunicación serie (RS232) también se gestionan las interrupciones mediante una rutina de atención a la interrupción. Se envían dos flags, uno que indica la recepción completa de la transferencia (ARM_USART_EVENT_RECEIVE_COMPLETE) y el otro que indica la completa transmisión (ARM_USART_EVENT_SEND_COMPLETE).

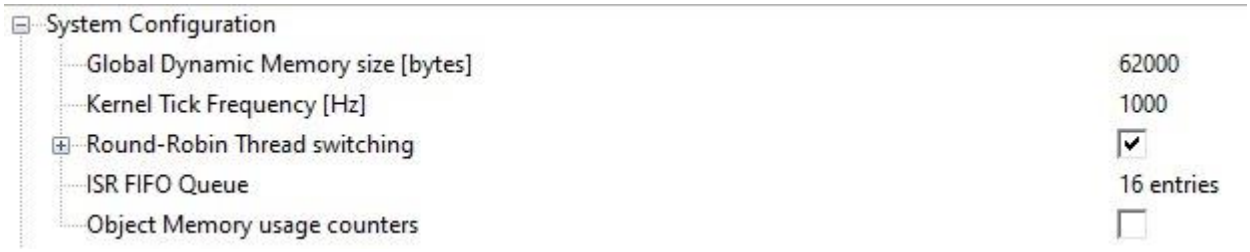
```
void myUSART_Callback(uint32_t event)
{
  uint32_t mask;

  mask = ARM_USART_EVENT_RECEIVE_COMPLETE  |
         ARM_USART_EVENT_TRANSFER_COMPLETE |
         ARM_USART_EVENT_SEND_COMPLETE     |
         ARM_USART_EVENT_TX_COMPLETE       ;

  if (event & ARM_USART_EVENT_RECEIVE_COMPLETE) {
    /* Success: Wakeup Thread */
    osThreadFlagsSet(tid_compc_rx, ARM_USART_EVENT_RECEIVE_COMPLETE);
  }

  if (event & ARM_USART_EVENT_SEND_COMPLETE) {
    /* Success: Wakeup Thread */
    osThreadFlagsSet(tid_compc_tx, ARM_USART_EVENT_SEND_COMPLETE);
  }
}
```

```
void myUART_Thread_rx(void* args)
{
  /*Initialize the USART driver */
  USARTdrv->Initialize(myUSART_Callback);
  /*Power up the USART peripheral */
  USARTdrv->PowerControl(ARM_POWER_FULL);
  /*Configure the USART to 115200 Bits/sec */
  USARTdrv->Control(ARM_USART_MODE_ASYNCHRONOUS |
                    ARM_USART_DATA_BITS_8 |
                    ARM_USART_PARITY_NONE |
                    ARM_USART_STOP_BITS_1 |
                    ARM_USART_FLOW_CONTROL_NONE, 115200);

  /* Enable Receiver and Transmitter lines */
  USARTdrv->Control (ARM_USART_CONTROL_TX, 1);
  USARTdrv->Control (ARM_USART_CONTROL_RX, 1);
```

Cada vez que se realiza tanto un Send como un Receive, se espera a ambos flags respectivamente:

```
USARTdrv->Send(&txMessage, sizeof(txMessage));          /* Get byte from UART */
osThreadFlagsWait(ARM_USART_EVENT_SEND_COMPLETE, osFlagsWaitAny, osWaitForever);
USARTdrv->Receive(&byte, 1);          /* Get byte from UART */
statusFlag = osThreadFlagsWait(ARM_USART_EVENT_RECEIVE_COMPLETE, osFlagsWaitAny, osWaitForever);
```

5. Aumento de la memoria total del sistema: para que no haya ningún problema con el funcionamiento del sistema por falta de memoria, se aumenta la memoria total del sistema operativo. Para ello, en el archivo RTX_Config.h, se fija el siguiente parámetro:



6. Inicialización de los hilos: se muestra la inicialización de los hilos del sistema, en el espacio necesario para ello

```
#ifdef RTE_CMSIS_RTOS2
  /* Initialize CMSIS-RTOS2 */
  osKernelInitialize ();

  /* Create thread functions that start executing,
  Example: osThreadNew(app_main, NULL, NULL); */
  Init_Principal ();
  Init_LCD ();
  Init_Hora ();
  Init_Joystick ();
  Init_MPU6050 ();
  Init_leds_N ();
  Init_com ();

  /* Start thread execution */
  osKernelStart();
#endif
```

4 DEPURACION Y TEST

4.1 Pruebas realizadas.

1. Módulo Hora (hora.c):

El objetivo de la prueba es verificar el correcto incremento de las variables “hh, mm, ss” de forma que implementen correctamente un reloj.

Condiciones de entrada:

- Inicialización correcta del hilo
- Inicialización correcta del timer que lleva a cabo la cuenta de la hora
- Observar que las variables almacenan el valor correcto

De esta forma se ejecuta el hilo observando en la ventana Watch estas variables. Prueba exitosa, obteniendo el funcionamiento esperado.

2. Módulo Joystick (joystick.c):

El objetivo de esta prueba es verificar el correcto funcionamiento del joystick en función de la pulsación realizada y su movimiento, además de determinar la duración de la pulsación. También se desea comprobar que la gestión de rebotes se realiza de forma correcta. Por último, se enviará por una cola un mensaje dependiendo del tipo de pulsación realizada.

Condiciones de entrada:

- Inicialización correcta del hilo
- Inicialización correcta de la cola de mensajes
- Inicialización de los timers que llevan la cuenta de los rebotes y la duración de la pulsación
- Detección del tipo de pulsación acorde a su gesto
- Observar en la ventana watch la variable de la cola de mensajes de forma que indique que pulsación se ha llevado a cabo
- Conexión correcta del joystick de la tarjeta mbed con la placa núcleo

Prueba exitosa, se consiguen suprimir los rebotes de las pulsaciones, así como observar el mensaje enviado por la cola dependiendo del gesto llevado a cabo en la pulsación (0x01: J_UP, 0x02: J_RGHT, 0x03: J_DOWN, 0x04: J_LEFT, 0x05: J_CNT_SHORT, 0x06: J_CNT_LONG

3. Módulo Acelerómetro (mpu6050.c)

El objetivo principal de esta prueba es verificar que cada segundo se recibe por la cola de mensajes los valores leídos de la aceleración y de la temperatura. Además, la aceleración y la temperatura debe ser acorde a la especificación y que por supuestos sea coherente.

Condiciones de entrada:

- Inicialización correcta del hilo
- Inicialización correcta de la cola de mensajes
- Inicialización correcta del bus I2C
- Comprobación del esclavo correcto (0x68)

- Configuración inicial del esclavo: despertar al esclavo, configuración correcta de los registros para la lectura.
- El esclavo devuelve, al menos, un valor.
- Interpretación correcta de la información leída.
- La aceleración varía al mover el sensor, dentro de los +/-2 g configurados.
- La temperatura aumenta a medida que calientas el sensor.
- Los mensajes se envían correctamente a través de la cola.

4. Módulo Leds (Leds_N.c)

El objetivo principal de esta prueba es verificar el correcto intercambio de estado de los leds de la tarjeta núcleo en función de los flags recibidos.

Condiciones de entrada:

- Inicialización correcta del hilo.
- Inicialización correcta del timer empleado para enviar los flags.
- Correcta variación del estado de los leds en función de los flags recibidos.

5. Módulo LCD (lcd.c)

Con esta prueba se pretende comprobar que el LCD gestiona correctamente la información que le llega por una cola de mensajes en función del modo en el que se encuentre, de forma precisa en el display:

Condiciones de entrada:

- Inicialización correcta del hilo
- Inicialización correcta de la cola de mensajes
- Inicialización correcta del bus SPI
- Observar en la pantalla la variación de la información dependiendo del modo en el que se encuentre
- Conexión correcta de la tarjeta mbed con la tarjeta núcleo

Prueba realizada de forma exitosa, se consigue ver la información en la pantalla del LCD de forma correcta dependiendo del mensaje enviado por la cola de mensajes del modo en el que se encuentra.

6. Módulo COMPC (compc.c)

Para realizar la prueba y depuración de dicho módulo, se ha creado un hilo de prueba, llamado principal.c, que simula lo que va a ocurrir en el diseño final, ya que el hilo principal necesita recibir y transmitir tramas. Para ello, desde compc.c inicializamos dos hilos: tid_compc_tx y tid_compc_rx. Uno se encargará de la transmisión de la trama y otro de la recepción. Además, se crean dos colas para transmitir las tramas, una para la transmisión y otra para la recepción.

Condiciones de entrada:

- Inicialización correcta de los hilos
- Inicialización correcta de las colas de mensajes
- Inicialización correcta del bus USART
- Observar en la ventana Watch Windows que las tramas se almacenan correctamente en los arrays con los bytes correctos
- Transmisión de la trama al programa TeraTerm y observar correctamente los datos enviados en su pantalla.

- Envío correcto de los archivos que contienen la trama desde el TeraTerm, y observar en la ventana Watch Windows que la trama recibida se almacena correctamente.
- Comprobación de que se envían tramas constantemente y no hay problema en la recepción.

A continuación, se muestra el proceso de como se ha probado dicho modo:

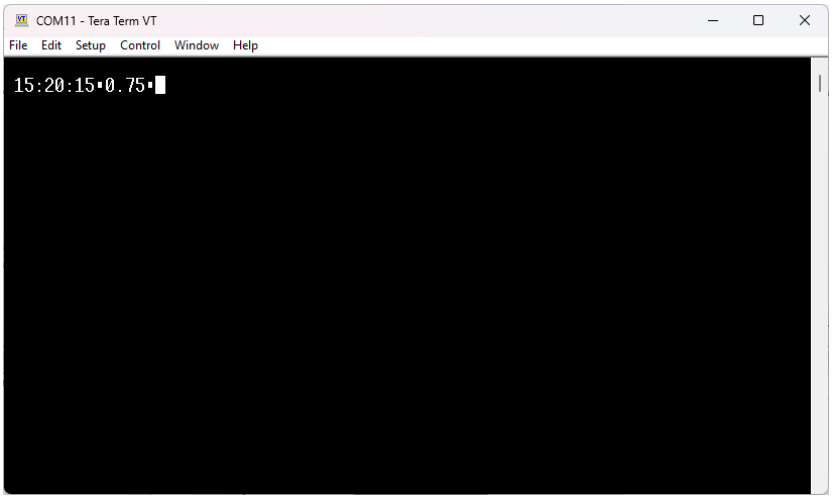
- Para probar que se transmiten tramas desde el principal al TeraTerm:

  Desde el hilo que simula al principal, se envían dos tramas: una que envía la trama de la hora y otra que envía trama que configura una referencia en el eje X, y se hace cada 5 s cada una:

```
void myUART_Thread_tx_test(void* args){

  static char txMessageF1[MAX_FRAME] = {0x01,0x20,0x0C,0x31,0x35,0x3A,0x32,0x30,0x3A,0x31,0x35,0xFE};
  static char txMessageF2[MAX_FRAME] = {0x01,0x25,0x08,0x30,0x2E,0x37,0x35,0xFE};

  osDelay(5000);
  osMessageQueuePut(idQueueTX (), &txMessageF1, 0U , 0U);
  osDelay(5000);
  osMessageQueuePut(idQueueTX (), &txMessageF2, 0U , 0U);

  while(1){
    osThreadYield();
  }
}
```

  Dicho put se recibe en el hilo compc.c, y se envía directamente con un Send al TeraTerm:

```
void myUART_Thread_tx(void* args){

  static char txMessage[MAX_FRAME];

  while(1){
    osMessageQueueGet(id_MsgQueue_TX, &txMessage, NULL, osWaitForever);

    //Se envia al ordenador
    USARTdrv->Send(&txMessage, sizeof(txMessage));          /* Get byte from UART */
    osThreadFlagsWait(ARM_USART_EVENT_SEND_COMPLETE, osFlagsWaitAny, osWaitForever);
  }
}
```



- Para probar que se reciben tramas desde el TeraTerm al principal:

  Desde el TeraTerm, en la pestaña de Control, se envían Macros y se recibe en el hilo del compc.c. Dicho hilo procesa la trama que le llega. Dicho procesamiento se realiza con una función, llamada processingFrame la cuál se activa cuando le llega el flag de la recepción del bus:

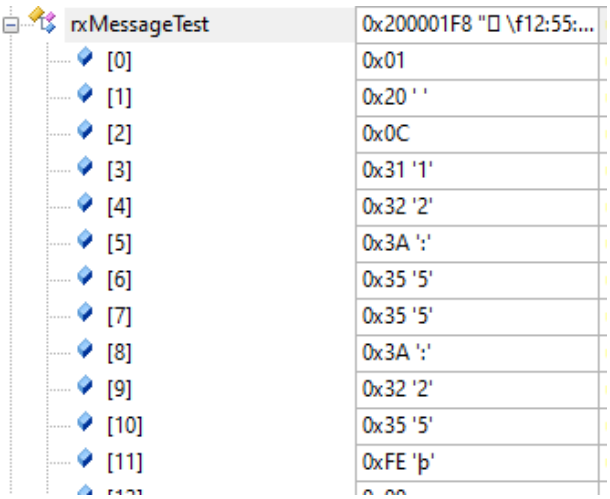
```
USARTdrv->Receive(&byte, 1);          /* Get byte from UART */
statusFlag = osThreadFlagsWait(ARM_USART_EVENT_RECEIVE_COMPLETE, osFlagsWaitAny, osWaitForever);
processingFrame(statusFlag);
```

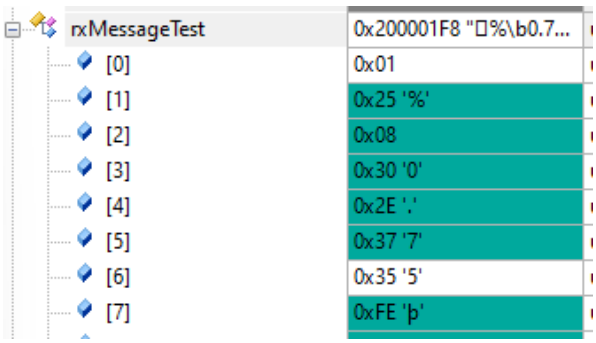
  En la siguiente función, se analiza la trama que le llega, byte a byte mediante un pequeño autómata de dos estados. Después, se realiza un put para enviar dicha trama al hilo principal:

```
void processingFrame (uint32_t flagRecepcion){

  if(flagRecepcion == ARM_USART_EVENT_RECEIVE_COMPLETE){
    if(estadoCom == InitState){
      if(byte == SOH){
        i = 0;
        estadoCom = DefaultState;
        frame[i] = byte;
        i++;
      }
    }else{
      if(byte != EOT){
        frame[i] = byte;
        i++;
      }else{
        estadoCom = InitState;
        frame[i] = byte; //Poner BP aqui

        if(frame[2] == strlen(frame)){
          osMessageQueuePut(id_MsgQueue_RX, &frame, 0U, 0U);
          memset(frame, 0x00, 50);
        }
      }
    }
  }
}
```

  El get se realiza en el principal, y para observar la trama recibida, se coloca un Breakpoint en el osDelay para poder observarla en la ventana Watch:

```
void myUART_Thread_rx_test(void* args){
  static char rxMessageTest[MAX_FRAME];

  while(1){
    osMessageQueueGet(idQueueRX (), &rxMessageTest, NULL, osWaitForever);
    osDelay(5000); //BP aqui para ver las tramas de recepción
  }
}
```





Y efectivamente, el array de recepción cada vez que recibe una nueva trama, se actualiza, borrando lo que sobre de las tramas anteriores.